Nurse Schedule Builder User Interface

Team Medecipher

CSCI 370: Section A

Pat Curran
Grant Josenhans
Collette Haberland
Daniel Ayers

INTRODUCTION

Client Background

Medecipher is a Denver-based IT health startup, funded by the National Science Foundation. Their flagship product is a scheduling program. Medecipher is working towards using machine learning to optimize nurse-to-patient ratios in hospitals. This concept is especially important in today's time, as the COVID-19 pandemic has led to a lot of early retirements within the healthcare industry. This has placed an increased strain on hospital staff as they have had to work more difficult schedules in order to meet patient needs. Medecipher's scheduling program forecasts hospital patient count and schedules nurses accordingly. Therefore, to fully utilize Medecipher's scheduler, a user interface is needed.

Project Goal

The purpose of our Field Session project is to design and create a client facing user interface. This UI allows clients to customize the nurse schedule, receive schedule data from an API, and save any modifications made to the schedule to a database. The majority of the backend was accomplished by Medecipher and previous CS@Mines' teams. Also, the most recent Mines' team created mock-up designs for the UI, refer to ***Figure 1***. Therefore, feedback was provided on each design from the Medecipher team. Taking into account these drafts and evaluations, the final UI design could incorporate any of these or be something completely new.

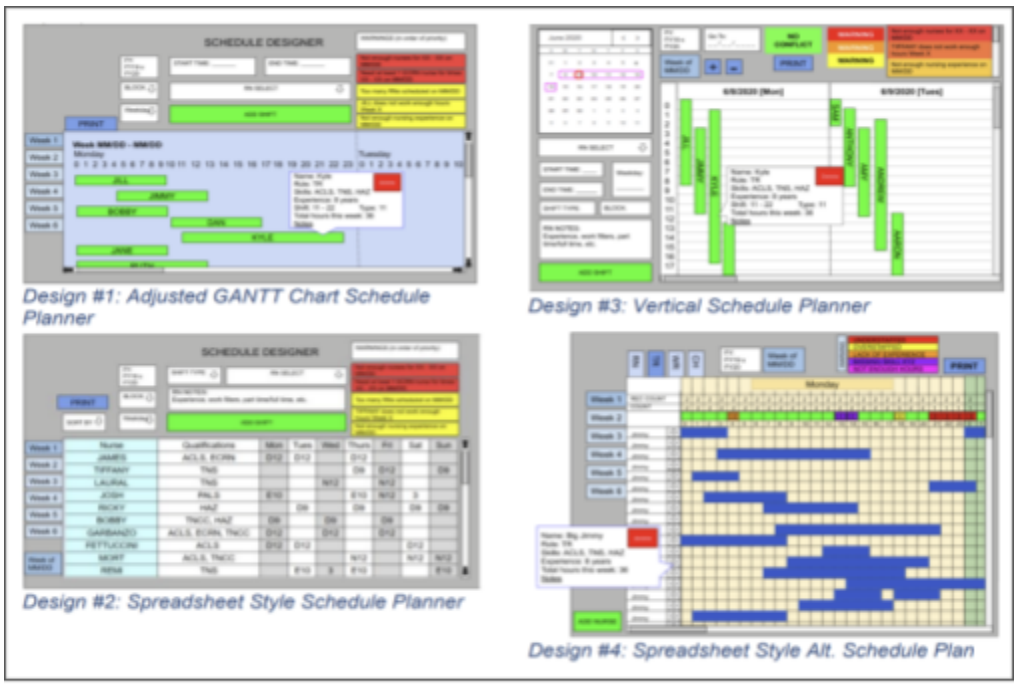


Figure 1: *design drafts from previous Mines' Field Session team*

REQUIREMENTS

Functional Requirements

Functional requirements include a UI layer and an API layer. The UI receives schedule data from the API layer and updates modifications made to the schedule. The UI also has the ability to make staffing recommendation choices. It displays a schedule view the user can interact with by choosing a date and recommendation type. The UI should send requests to a REST API. This API pulls back and caches data, returning data points and a schedule that displays recommendations. The UI also provides warnings related to sparsity and nurse burn-out. For example, if a warning should be shown if too few nurses are staffed on a particular floor. These thresholds will be adjusted to the needs of a given client.

The API layer abstracts out communication with the API. This layer will connect between the UI, the data store, and the microservices/data science layer that has previously been developed. Unit tests may be conducted with API calls and end-to-end tests will be performed as a user.

Non-Functional Requirements

The non-functional requirements include the operating system specified by Medecipher. Linux/Windows will be used for the UI, and Linux for the API. The majority of tasks will be completed locally and won't require much processing power. Additional requirements will mainly be handled through Amazon Web Services. These include the size of clusters, elastic costs, authentication/security, etc. The main requirement is an automated, simple visual that follows standard coding practices. Regular meetings with Medecipher allowed us to experiment with different visual designs that they can provide feedback on, creating a cycle of design and refinement.

SYSTEM ARCHITECTURE

High Level Design

The main component involves a UI that is deployed through Amazon Web Services. The UI is a shift scheduler for nurses, which includes a weekly calendar, shift warnings, alternating/updating existing and new shifts, and nurse details. This UI connects to the service layer that handles the business logic. The business logic notifies the user about various potential warnings and details within each shift. The service layer is connected to the API, which is in existence. The API simply calls various functions to the existing database. The database dynamically updates based on the scheduler UI. All of the existing architecture includes Kubernetes/Micro services, stream processor, data science, ETL/data processing. This back-end layer already connects to the API and handles the actual scheduling. All connections between components are displayed in ***Figure 2***.

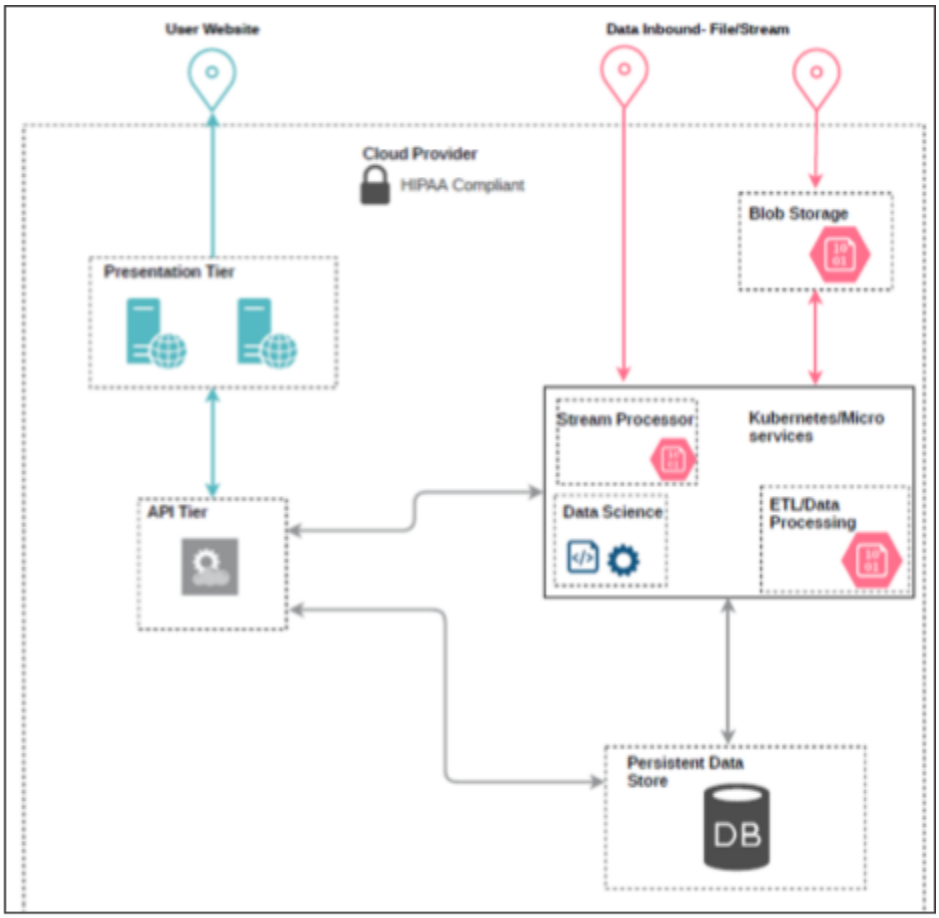


Figure 2: *High-level design, provided by Medecipher*

Design Details

The scheduler UI was implemented in Angular, a component-based platform that favors structure and self-containment. Its organization is similar to object-oriented programming by allowing for components to be given hierarchy for the sake of cleanliness. The UI's architecture can be seen in the UML diagram below (***Figure 3***). The parent 'app-component' class allows for all other components to be instantiated. We chose to create a 'sidebar' component that would hold all buttons/tables along the left side of the UI. The 'button' component implements the 'nurse-dropdown' component to display a pop-up menu when a button is clicked. Placing the dropdown information in its own component as opposed to storing it in the button component allowed for greater readability and made working directly with the dropdown simpler.

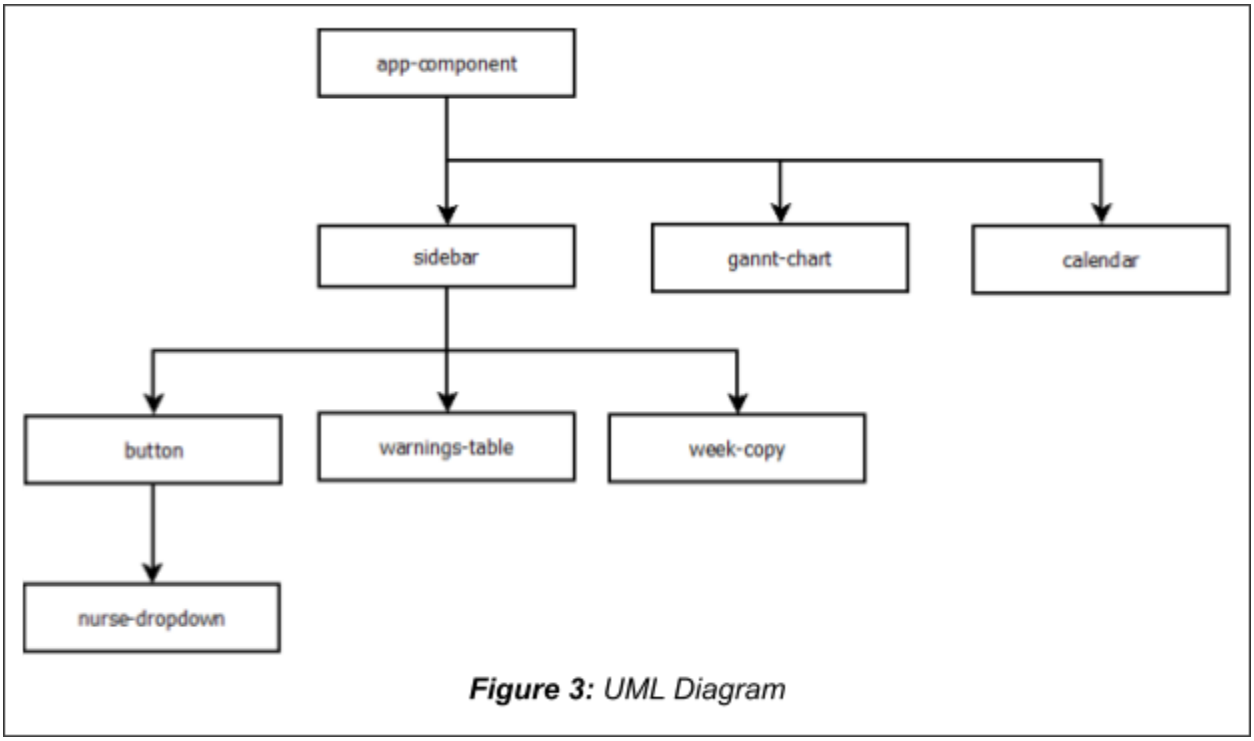


Figure 3: *UML Diagram*

TECHNICAL DESIGN

Acquiring The Data

The main component of the UI is the schedule itself. In order to populate this component, the data needed to be read in from a .csv file, provided to us by Medecipher with stripped information for security purposes. The actual API is still being worked on by Medecipher, to acquire actual HIPAA data from cloud storage. After research, testing, and planning, the best direction to take was using an Angular service.

Angular services are the logic that are used to perform specific tasks. They let one define code or functionalities that are then accessible and reusable in many other components within the Angular project. Services are similar to an interface in Java. Since Angular revolves around everything being in their own component, these components can delegate certain tasks to services. This results in an injectable service class, enabled through dependency injections.

Dependency injection is a programming paradigm that makes a class independent of its dependencies. Dependency injection enables the creation of dependent objects outside of a class, while providing those very objects to a class in numerous ways. The flow is shown in ***Figure 4***.

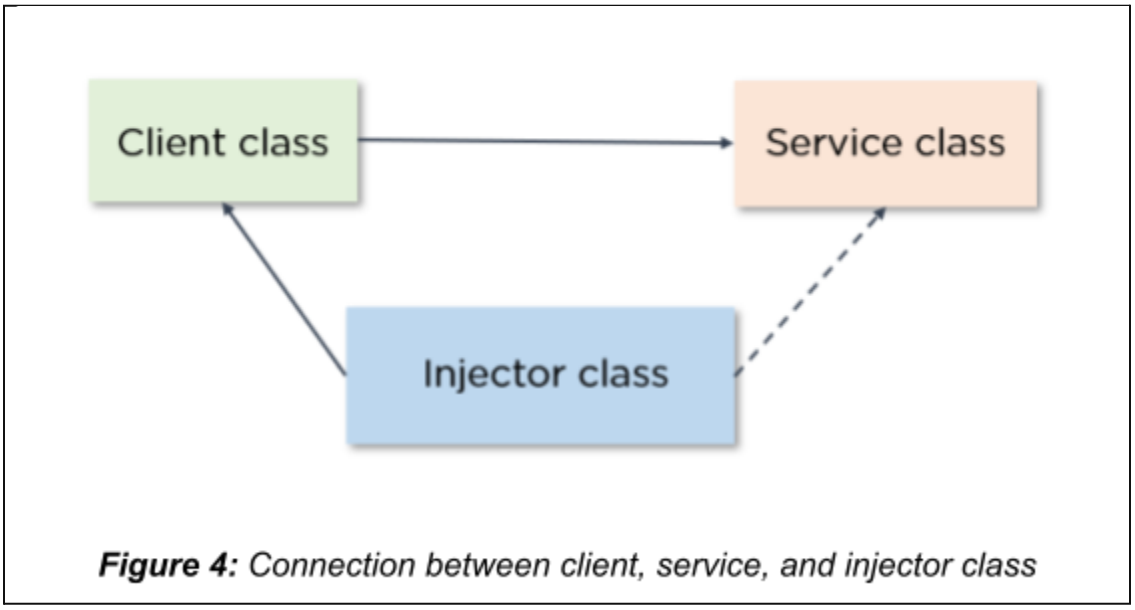


Figure 4: *Connection between client, service, and injector class*

We used the HTTP client in Angular to subscribe to the .csv data. This caused the data to be read in asynchronously, which makes it difficult to use in subsequent logic. While other aspects of the code were relying on the array populated from the .csv data, at the same time, the data was being read. Therefore, this led to an empty array being used throughout our code. Our final solution resulted in instantiating a new service for every function that needs the data. This allowed the flow of acquiring the data, then using and modifying it in any way to be in order.

Connecting Calendar Data to Other Components

Our Angular project is broken down into many components. Each represents a different aspect of the UI. Every component revolves around the data that is selected from the calendar component. Specifically, everytime a new date is selected, the schedule needs to populate accurate data, the add and delete shift buttons need to update .csv, and the copy schedule button shows the Sunday of that week.

Originally when researching and planning how to pass this date data between components, the sequence was thought to be:

1. Event triggers when a new date is selected on: store the new date in variable
2. Pass that variable to app-component (parent of all components)
3. All other parents can grab the updated date/variable from app-component and use as they wish

However, the outline was found out to not be as simple as originally thought. In Angular, there is a certain hierarchy when trying to pass data between different components. The use of decorators @Input and @Output allows data to be shared in Angular. ***Figure 5*** below depicts the most simple of cases: passing data only between two components: the child and the parent.

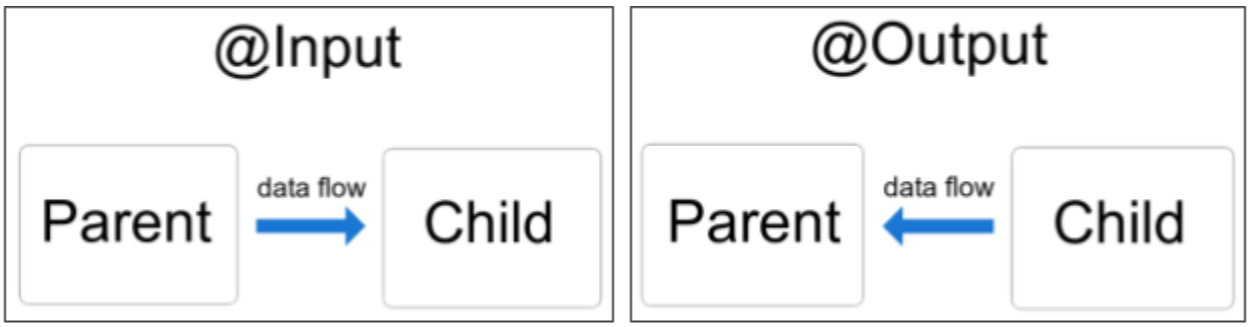


Figure 5: *Data flow between parent and child component, using decorators*

Figure 6 shows a specific example of passing the date from the calendar-component to app-component to gantt-chart-component. Child 2 can be replaced with any other child component of app-component, since the parent already has the necessary information.

Child 1: calendar-component:

calendar-component.ts

```
  onChange(args: any): any{
    this.today = args.value;
    this.item = args.value;
    this.messageEvent.emit(this.item)
  }

  item = this.today;
  @Output() messageEvent = new EventEmitter<any>();
}
```

calendar-component.html

```
<ejs-calendar [value]='today' (renderDayCell)='disabledDate($event)' (change)='onChange($event)'></ejs-calendar>
</div>
```

Parent: app-component:

app-componet.ts

```
  receiveMessage($event:any){
    this.date = $event;
    this.parentMessage = this.date;
  }
```

app-component.html

```
<div class='page_container'>
    <div class='sidebar'>
        <app-calendar (messageEvent)="receiveMessage($event)"></app-calendar>
        <app-sidebar [interMessage]="parentMessage"></app-sidebar>
    </div>

    <div class='gannt-chart'>
        <app-gannt-chart [childMessage]="parentMessage"></app-gannt-chart>
    </div>

</div>
```

Child 2: gantt-chart-component

gantt-chart-componet.ts

```
@Input() childMessage: any | undefined;
ngOnChanges(changes: any){
  this.ngOnInit()
}
```

Figure 6: *code for date being passed to generate updated schedule*

QUALITY ASSURANCE

- **User Interface Testing**
  - Tested edge cases
    - Added incomplete shifts
    - Calendar dates far in the future/past
  - Tested standard functionality
    - Read .csv schedule files
  - Verify Medecipher is satisfied with the final layout

- **Code Reviews**
  - 2 other members of team reviewed code before merging
  - Gave Medecipher access to our GitHub repository

- **User Acceptance testing**
  - Is Medecipher content with the product?
  - Are nurses/floor managers content with the product?

- **Code Metrics**
  - Used standard design patterns/components most commonly associated with framework in use (Angular10)
    - Component based architecture
    - Services/injectables
    - etc.

- **Static Analysis**
  - Pair programed to ensure correctness and ‘sanity checks’
  - Refactored and commented code for legibility

- **Dynamic Analysis**
  - Continuous delivery pipeline for automated testing
  - End-to-end tested similarly except as user

Ethics

- **Pertinent principles**
  - General: contribute to society and to human well-being, acknowledging that all people are stakeholders in computing
  - Professional: accept appropriate professional review.
  - Self: learned web development and practiced good software engineering principles

- **Most danger of violation**
  - Public: kept patient data secure
  - Product: created a quality product that meets Medecipher’s standards

RESULTS

Goals Achieved

For our project, we were able to implement a visually-appealing nurse scheduling tool that reads in data from a .csv file to display a schedule to a user. The UI for this scheduler is easy to read and displays pertinent information in an accessible way. This is done by sorting nurse shifts by type of nurse to show how many nurses of each type are working at a given time. In addition, the ‘Add Shift’ button filters nurse IDs by type of nurse to make finding a particular nurse simpler for the user. Finally, the schedule data for a particular day can be shown by clicking on a calendar in the upper corner of the screen.

The schedule has the added functionality of grouping nurses together by type (i.e. CRN, EDRN. Organizing the schedule by type of nurse was requested by Medecipher because floor managers need to know what nurses are scheduled for a given time. Grouping like nurses together allows this information to be more readily accessible and creates a more organized schedule.

In addition, the schedule can also be scrolled independently from the rest of the page. This allows the schedule to be navigated and read without disrupting the rest of the webpage and improves the user experience while using our web application.

Another goal we achieved was the ability to add shifts to a given schedule. By clicking a button the user is presented with dropdown menus for nurse type, shift type, and nurse ID. Nurse ID is filtered based on the selection for nurse type, and shift type is limited to the types of shifts specified in the .csv file. This allows the user to select from a list of presets instead of needing to manually select start and end times for a given shift. If the user attempts to add a shift without filling out all fields they will be prompted with an alert to choose an option from the remaining dropdowns.

The calendar component was fully implemented through an open source package, syncfusion. Since the optimized nurse schedule optimizes shifts 4 weeks at a time, we disabled all months before and after the current month. This prevents the user from modifying a schedule where the data has not been supplied yet. Also, by triggering an event emitter everytime the user selected a new date, we sent that updated date variable to the other components that needed that information as well. This way, the gantt-chart, copy schedule button updates dynamically with the calendar.

Unimplemented Features

We were unable to implement an API layer to transfer data from our UI to an external database using AWS. Our project is currently limited to reading and writing data to and from a .csv file. This allowed us to prototype a layout for the UI and implement data processing. However, as of now the scheduler is unable to connect to Medecipher's database for handling proper schedule data.

We were also unable to implement the ability to delete shifts from the schedule. Our intention was to create a popup when a user clicks on a shift. This popup would display information about the shift to the user (i.e. nurse experience, name, etc.). It would also have a button to remove the shift from the schedule, which would do so after double checking if the user wanted to delete that shift. This functionality was left unimplemented in our final product.

The copy button was another feature we were unable to implement. The copy button was intended to copy the schedule for a given week forward by the number of weeks the user specifies. This would allow the user to propagate a schedule that they like and that works well for the nurses into future weeks without having to manually add/delete shifts to match the desired schedule. Additionally, we wanted to have this button calculate warnings for these future weeks in case the schedule is no longer ideal because of patient flow or other changes causing new warnings to arise, but this is also unimplemented. Currently, the copy button in our work is a placeholder that prompts the user to input the number of weeks they would like to copy a schedule. It does not have any further functionality.

Finally, we ran out of time to fully implement the warning table in our work. The table we created stores hardcoded values for the number of shifts that are understaffed, overstaffed, how many nurses are overworked, and how many shifts have too little experience. We included basic functionality for the color of each row, making a row red if the warning is present and green if it is not. This makes it easy to see if there are no errors because there will be four green boxes. The table is disconnected from the schedule and cannot update based on the selected schedule information at this time.

FUTURE WORK

In the future, our work could be extended by implementing features we did not have time to address. This includes the delete shift button, the warnings table, and the copy shift button among others. Data transfer is one of the most important next steps for the scheduler. Currently, the add shift button has no way to properly create a shift and update the .csv file accordingly. In addition, the scheduler only works with local data from a .csv. The software is disconnected from Medecipher's API and can only be used to display our test data. The software we created has several ways to expand and grow in utility through new features and improvements.

Our final steps before submitting our work to Medecipher include writing detailed documentation and another pass of refactoring over our final work. We currently have several comments explaining our work and the way we implemented certain features. However, we want to ensure that our project is as clear and easy to understand as possible for the next CS@Mines team that will take over from where we left off. This means we will include documents like a detailed 'readme' that explains each component of our Angular architecture and how they relate to one another. We will also include more comments throughout our .ts and .html files to explain our work in greater depth. We realize that it is easy to understand our own thought processes and ideas because we have been familiar with them for so much of the semester. Creating in-depth documentation will ensure that whoever continues our work will be able to understand those ideas and be able to build off of them more effectively.

LESSONS LEARNED

We learned a lot over the semester, including how to use angular, CSS, HTML, and typescript. While using angular we discovered that angular's component-based architecture is very similar to object oriented programming. Each component in angular can be thought of as an individual 'class' that stores its own information. For example, a 'warning table' component can fully encapsulate all of the information about the warning table. These components can then be arranged in the app-component.html file, which serves to organize each of the components. However, instead of using a single object oriented programming language, angular packages 3 languages together (CSS, HTML, typescript). This created a steep learning curve at the start of the semester when we first began working on the project. However, as we grew more familiar with the architecture we were able to work more efficiently and achieve further progress towards our goals each week.

Another of the lessons we learned was the importance of clear, open communication. Throughout the semester, we met with Medecipher every other week to give an update on our progress and receive feedback on our work. This was incredibly helpful as it allowed us to pivot and redesign our app throughout the semester, rather than having a massive set of changes to implement near the end of the project. It also allowed us to learn a deeper understanding of the project requirements. This was especially true towards the beginning of the semester when we did not have a firm grasp on what we were supposed to accomplish. Having frequent communication with Medecipher was vital in helping us understand our project and gradually fine-tune it to their needs.

Finally, we also found that the agile framework greatly helps with adaptability. Within this framework, we chose to work on one component/piece of functionality at a time. This allowed us to work in depth on a few select components and make them work well as opposed to having a large number of features that are each limited. In addition, it allowed us to more easily pivot when an idea didn't quite work or when we needed to readjust a component. Working within the agile framework helped make our code more fluid and open to tweaks and changes throughout the semester.

TEAM PROFILE

Collette Haberland

Hi, my name is Collette Haberland, and I am currently a senior at Mines, graduating in spring 2022 with a Bachelor's in Computer Science and focus in Data Science. I am from Castle Pines, CO. I am involved in a lot of things at Mines, such as Alpha Phi, PATHS, Tau Beta Pi, Order of Omega, Intramural and Club Sports Marketing Executive, and more! I will be moving to Seattle, when I graduate, to work full-time for Nordstrom as a software engineer!

Role: creation of calendar component and sharing selected date between components

Daniel Ayers

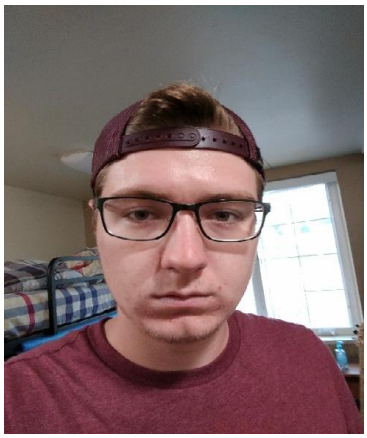
My name is Daniel Ayers. I am a senior in Computer Science pursuing a minor in Electrical Engineering. I intend to earn a Master's degree in Computer Science with a focus in algorithmic robotics. I enjoy rock climbing, reading, and playing classical guitar.

Role: creation of add and delete shift buttons, helped resolve asynchronous data read-in.

Grant Josenhans

My name is Grant Josenhans, and I am a Senior in Computer Science here at Mines. I am currently taking courses in order to receive the Mines Cyber Defense Education Certificate, and once I graduate after this school year I will get a job in cyber security. I like reading fantasy novels, solving rubik's cubes, playing video games, and rock climbing. I don't have pictures of myself, so I used the profile pic I took freshman year for Canvas.

Role: Displaying current week based on the date of the current week's Sunday. Creation of copy week button and addition of some functionality to the warning's table.



Patrick Curran

My name is Patrick Curran, and I am a junior at Colorado School of Mines studying Computer Science. When I am not grinding hard at school I enjoy watching and arguing about boxing. I love seeing live music all over the country and have seen over 100 shows from my favorite band, The Disco Biscuits. My wife and I spend summers camping around the country.

Role: created Nurse Schedule Service, designed the schedule GANNT chart, also some comedic relief

APPENDICES

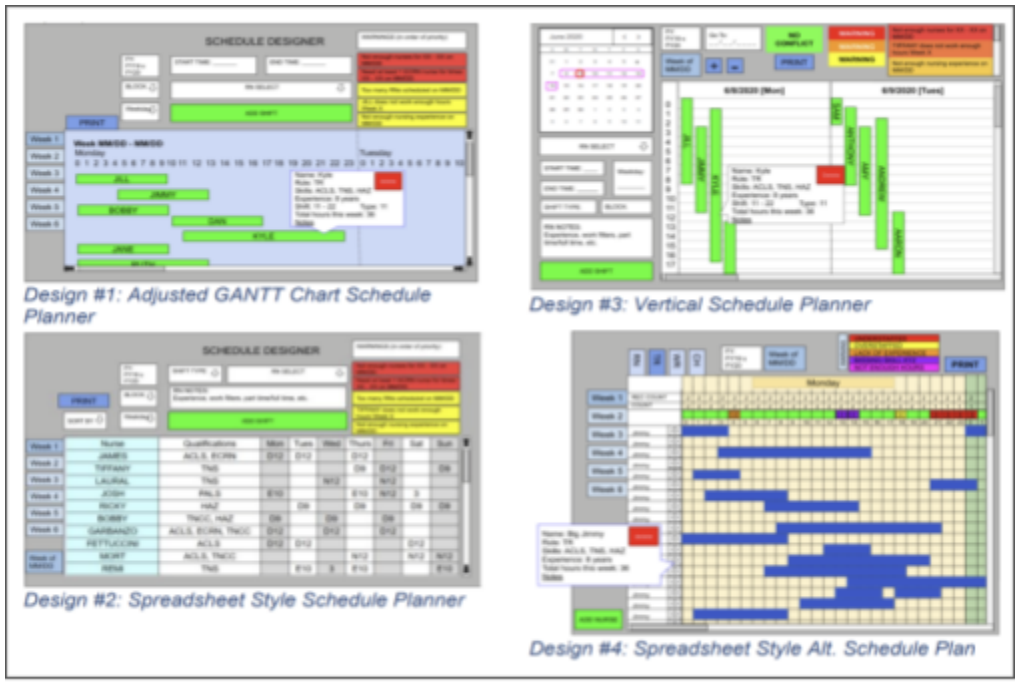


Figure 1: *design drafts from previous Mines' Field Session team*

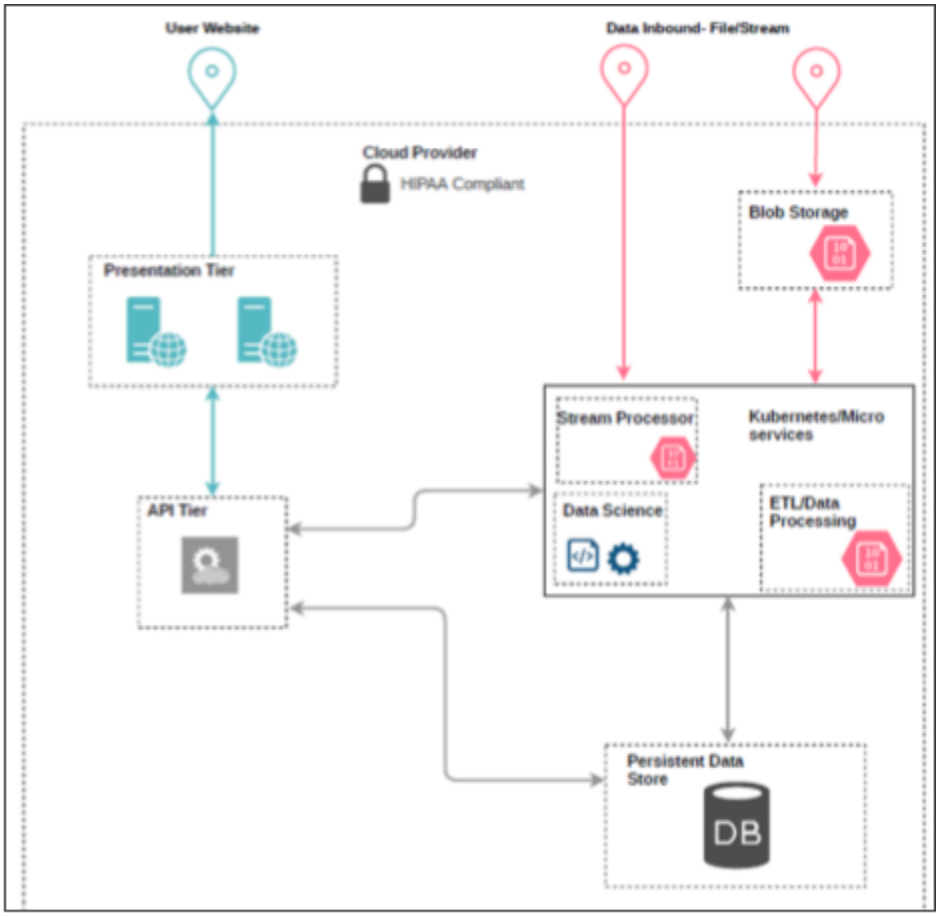


Figure 2: *High-level design, provided by Medecipher*

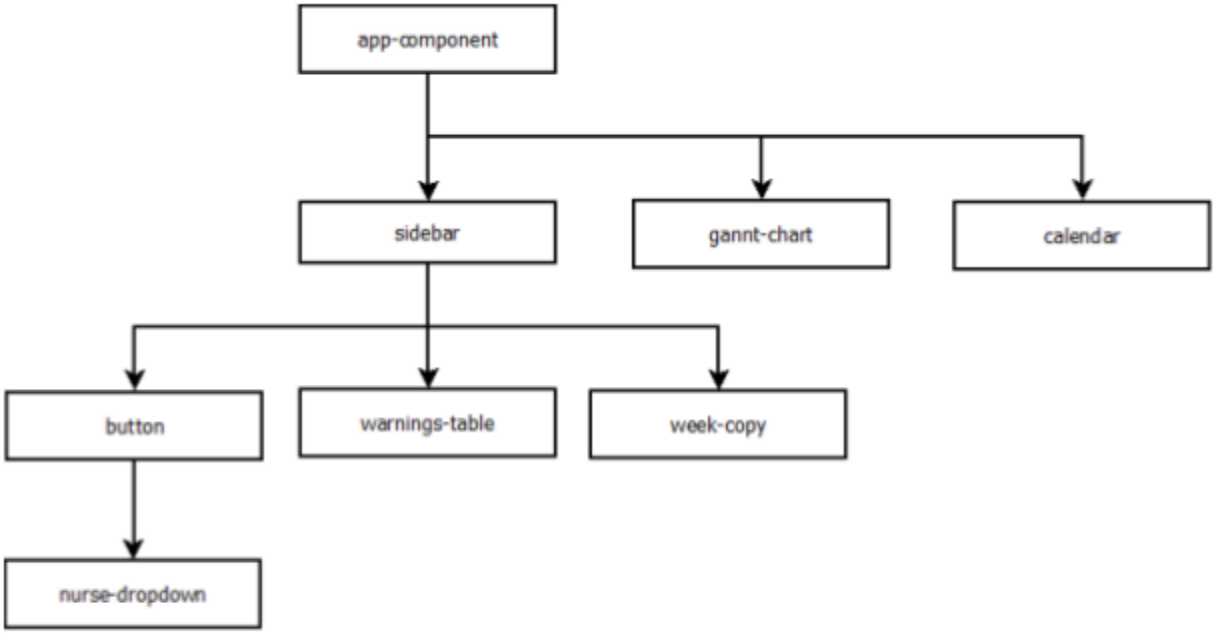


Figure 3: *UML Diagram*

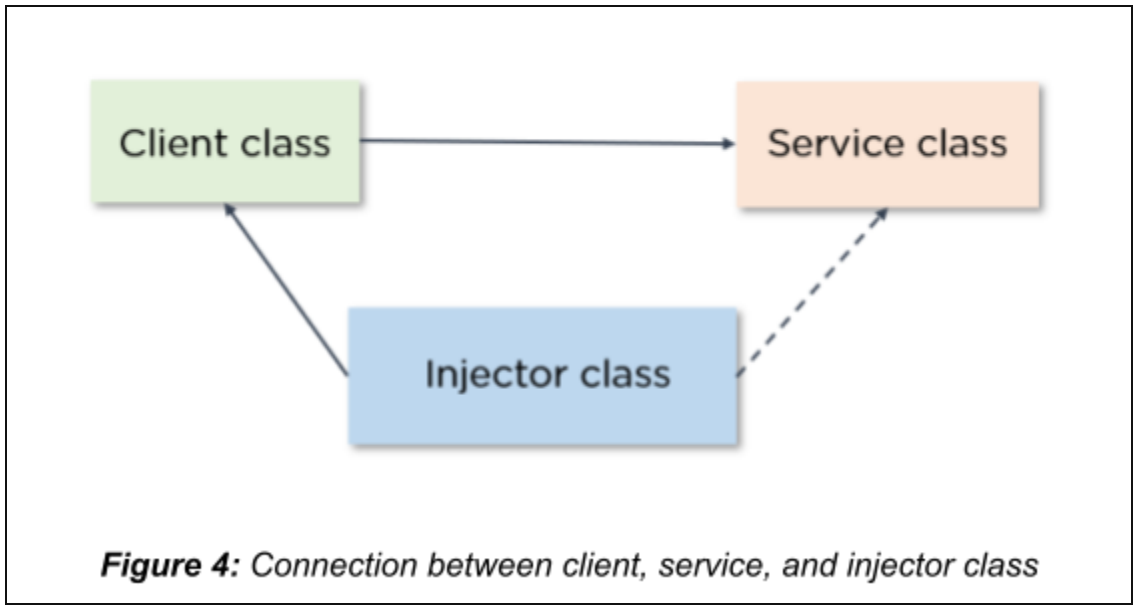


Figure 4: *Connection between client, service, and injector class*

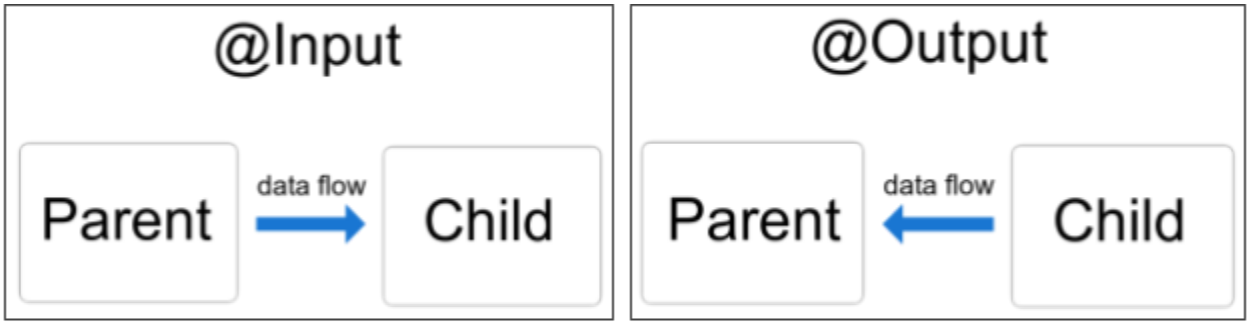


Figure 5: *Data flow between parent and child component, using decorators*

Child 1: calendar-component:

calendar-component.ts

```
  onChange(args: any): any{
    this.today = args.value;
    this.item = args.value;
    this.messageEvent.emit(this.item)
  }

  item = this.today;
  @Output() messageEvent = new EventEmitter<any>();
}
```

calendar-component.html

```
<ejs-calendar [value]='today' (renderDayCell)='disabledDate($event)' (change)='onChange($event)'></ejs-calendar>
</div>
```

Parent: app-component:

app-componet.ts

```
  receiveMessage($event:any){
    this.date = $event;
    this.parentMessage = this.date;
  }
```

app-component.html

```
<div class='page_container'>
    <div class='sidebar'>
        <app-calendar (messageEvent)="receiveMessage($event)"></app-calendar>
        <app-sidebar [interMessage]="parentMessage"></app-sidebar>
    </div>

    <div class='gannt-chart'>
        <app-gannt-chart [childMessage]="parentMessage"></app-gannt-chart>
    </div>

</div>
```

Child 2: gantt-chart-component

gantt-chart-componet.ts

```
@Input() childMessage: any | undefined;
ngOnChanges(changes: any){
  this.ngOnInit()
}
```

Figure 6: *code for date being passed to generate updated schedule*